

Section 2: Heaps and Asymptotics

0. Big-Oh Proofs

For each of the following, prove that $f \in \mathcal{O}(g)$.

(a) $f(n) = 7n$ $g(n) = \frac{n}{10}$

(b) $f(n) = 1000$ $g(n) = 3n^3$

(c) $f(n) = 7n^2 + 3n$ $g(n) = n^4$

(d) $f(n) = n + 2n \lg n$ $g(n) = n \lg n$

1. Look Before You Heap

(a) Insert 10, 7, 15, 17, 12, 20, 6, 32 into a *min heap*.

(b) Now, insert the same values into a *max heap*.

(c) Now, insert 10, 7, 15, 17, 12, 20, 6, 32 into a *min heap*, but use Floyd's buildHeap algorithm.

(d) Insert 1, 0, 1, 1, 0 into a *min heap*.

2. \mathcal{O} My God!

Recall the definition of $f \in \Omega(g)$ is as follows:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). f(n) \geq cg(n)$$

Prove that $4n^2 + n^5 \in \Omega(n)$.

3. Not to Tree

For the following code snippet, find a recurrence for the worst case runtime of the function, and then find a closed form for the recurrence.

Consider the function f :

```
1 f(n) {  
2   if (n <= 0) {  
3     return 1;  
4   }  
5   return 2 * f(n - 1) + 1;  
6 }
```

- Find a recurrence for $f(n)$.

- Find a closed form for $f(n)$.

4. To Tree

Consider the function h :

```
1 h(n) {
2   if (n <= 1) {
3     return 1
4   } else {
5     return h(n/2) + n + 2*h(n/2)
6   }
7 }
```

(a) Find a recurrence $T(n)$ modeling the *worst-case runtime complexity* of $h(n)$.

(b) Find a closed form to your answer for (a).

5. To Tree or Not to Tree

Consider the function f . Find a recurrence modeling the worst-case runtime of this function and then find a Big-Oh bound for this recurrence.

```
1 f(n) {
2   if (n == 0) {
3     return 0
4   }
5   int result = f(n/2)
6   for (int i = 0; i < n; i++) {
7     result *= 4
8   }
9   return result + f(n/2)
10 }
```

(a) Find a recurrence $T(n)$ modeling the worst-case time complexity of $f(n)$.

(b) Find a closed form for $f(n)$

Extra Practice On Following Pages

6. Is Your Program Running? Better Catch It!

For each of the following, determine the tight $\Theta(\cdot)$ bound for the worst-case runtime in terms of the free variables of the code snippets.

(a)

```
1 int x = 0
2 for (int i = n; i >= 0; i--) {
3     if ((i % 3) == 0) {
4         break
5     }
6     else {
7         x += n
8     }
9 }
```

(b)

```
1 int x = 0
2 for (int i = 0; i < n; i++) {
3     for (int j = 0; j < (n * n / 3); j++) {
4         x += j
5     }
6 }
```

(c)

```
1 int x = 0
2 for (int i = 0; i < n; i++) {
3     for (int j = 0; j < i; j++) {
4         x += j
5     }
6 }
```

(d)

```
1 int x = 0
2 for (int i = 0; i < n; i++) {
3     if (n < 1000000) {
4         for (int j = 0; j < i * i * n; j++) {
5             x += 1
6         }
7     } else {
8         x += 1
9     }
10 }
```

(e)

```
1 int x = 0
2 for (int i = 0; i < n; i++) {
3     if (i % 5 == 0) {
4         for (int j = 0; j < n; j++) {
5             if (i == j) {
6                 for (int k = 0; k < n; k++) {
7                     x += i * j * k
8                 }
9             }
10         }
11     }
12 }
```

7. Asymptotics Analysis

Consider the following method which finds the number of unique Strings within a given array of length n .

```
1 int numUnique(String[] values) {
2     boolean[] visited = new boolean[values.length]
3     for (int i = 0; i < values.length; i++) {
4         visited[i] = false
5     }
6     int out = 0
7     for (int i = 0; i < values.length; i++) {
8         if (!visited[i]) {
9             out += 1
10            for (int j = i; j < values.length; j++) {
11                if (values[i].equals(values[j])) {
12                    visited[j] = true
13                }
14            }
15        }
16    }
17    return out;
18 }
```

Determine the tight $\mathcal{O}(\cdot)$, $\Omega(\cdot)$, and $\Theta(\cdot)$ bounds of each function below. If there is no $\Theta(\cdot)$ bound, explain why. Start by (1) constructing an equation that models each function then (2) simplifying and finding a closed form.

(a) $f(n)$ = the worst-case runtime of `numUnique`

(b) $g(n)$ = the best-case runtime of `numUnique`

(c) $h(n)$ = the amount of memory used by `numUnique` (the space complexity)

8. Oh Snap!

For each question below, explain what's wrong with the provided answer. The problem might be the reasoning, the conclusion, or both!

(a) Determine the tight $\Theta(\cdot)$ bound for the worst-case runtime of the following piece of code:

```
1 public static int waddup(int n) {
2     if (n > 10000) {
3         return n
4     } else {
5         for (int i = 0; i < n; i++) {
6             System.out.println("It's dat boi!")
7         }
8         return 0
9     }
10 }
```

Bad answer: The runtime of this function is $\mathcal{O}(n)$, because when searching for an upper bound, we always analyze the code branch with the highest runtime. We see the first branch is $\mathcal{O}(1)$, but the second branch is $\mathcal{O}(n)$.

(b) Determine the tight $\Theta(\cdot)$ worst-case runtime of the following piece of code:

```
1 public static void trick(int n) {
2     for (int i = 1; i < Math.pow(2, n); i *= 2) {
3         for (int j = 0; j < n; j++) {
4             System.out.println("(" + i + "," + j + ")")
5         }
6     }
7 }
```

Bad answer: The runtime of this function is $\mathcal{O}(n^2)$, because the outer loop is conditioned on an expression with n and so is the inner loop.

9. Big-Of Bounds

Consider the function f . Find a recurrence modeling the worst-case runtime of this function and then find a Big-Oh bound for this recurrence.

```
1 f(n) {
2   if (n == 0) {
3     return 0
4   }
5
6   int result = 0
7   for (int i = 0; i < n; i++) {
8     for (int j = 0; j < i; j++) {
9       result += j
10
11     }
12   }
13   return f(n/2) + result + f(n/2)
14 }
```

(a) Find a recurrence $T(n)$ modeling the worst-case time complexity of $f(n)$.

(b) Find a Big-Oh bound for your recurrence.

10. Odds Not in Your Favor

Consider the function g . Find a recurrence modeling the worst-case runtime of this function, and then find a closed form for the recurrence.

```
1 g(n) {
2   if (n <= 1) {
3     return 1000
4   }
5   if (g(n/3) > 5) {
6     for (int i = 0; i < n; i++) {
7       println("Yay!")
8     }
9     return 5 * g(n/3)
10  }
11  else {
12    for (int i = 0; i < n * n; i++) {
13      println("Yay!")
14    }
15    return 4 * g(n/3)
16  }
17 }
```

(a) Find a recurrence $T(n)$ modeling the worst-case time complexity of $g(n)$.

(b) Find a closed form for the above recurrence.